

Tugas Besar 2 IF3170 Inteligensi Buatan
Implementasi Algoritma Pembelajaran Mesin

Diajukan untuk memenuhi tugas mata kuliah IF3170 Inteligensi Buatan



Disusun oleh Kelompok 26:

Maggie Zeta RS	13521117
Shafiq Irvansyah	13522003
Devinzen	13522064
Rayendra Althaf TN	13522107
Chelvadinda	13522154

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI.....	2
BAB 1	
Penjelasan Implementasi Program.....	3
1. Implementasi Algoritma KNN.....	3
2. Implementasi Algoritma Naive Bayes.....	4
3. Implementasi Algoritma ID3.....	7
BAB 2	
Tahap Cleaning dan Preprocessing.....	12
1. Data Cleaning.....	12
2. Data Preprocessing.....	12
3. Pipeline Preprocessing.....	13
BAB 3	
Hasil.....	14
1. Hasil Algoritma Naive Bayes.....	14
2. Hasil Algoritma ID3.....	14
3. Hasil Algoritma KNN.....	15
KONTRIBUSI.....	16
REFERENSI.....	17

BAB 1

Penjelasan Implementasi Program

1. Implementasi Algoritma KNN

K-Nearest Neighbor (KNN) adalah algoritma berbasis *instance* yang menggunakan pendekatan *lazy learning*. Algoritma ini menyimpan semua data train dan melakukan prediksi data *test* berdasarkan kemiripan (*distance*) dengan *K-Nearest Neighbor*.

Implementasi:

1. *Distance*

Kami menggunakan *Minkowski Distance*:

- $p=1$: *Manhattan*
- $p=2$: *Euclidean*

Rumus *Minkowski Distance*:

$$d(A, B) = \left(\sum_{i=1}^n |A_i - B_i|^p \right)^{\frac{1}{p}}$$

2. Prediksi

- Menghitung jarak semua data *test* ke data *train*
- Menggunakan mayoritas *class* dari *K-Nearest Neighbor* menggunakan Counter

3. Implementasi *From Scratch*

Kami menggunakan *class* `KNN_FromScratch` yang menyimpan data *train* dan melakukan prediksi *class* data *test*.

- `fit()`: untuk menyimpan data *train*
- `predict()`: untuk memprediksi *class* data *test*

```
import numpy as np
from collections import Counter
import pickle

def distance(A, B, p):
    # A dan B array point, p parameter (lihat
    __init__)
    return np.sum(np.abs(A - B) ** p) ** (1 / p)

class KNN_FromScratch:
    def __init__(self, n_neighbors: int, p: float):
        # p: parameter angka untuk menghitung
        distance Minkowski
        # 1 untuk manhattan, 2 untuk euclidean,
        minkowski terserah mau masukin angka berapa (jangan
        yg aneh aneh)
```

```

        self.n_neighbors = n_neighbors
        self.p = p

    def fit(self, X_train, Y_train):
        # asumsi scaling dan encoding sudah di
notebook
        # fungsi ini cuma simpan training data
        self.X_train = X_train.to_numpy()
        self.Y_train = Y_train.to_numpy()
        return

    def predict_instance(self, instance):
        distances = [distance(i, instance.to_numpy(),
self.p) for i in self.X_train]
        nearest_neighbors_index =
np.argsort(distances)[:self.n_neighbors] # argsort yg
direturn indexnya
        nearest_neighbors =
self.Y_train[nearest_neighbors_index]
        nearest_neighbors = [label.item() if
isinstance(label, np.ndarray) else label for label in
nearest_neighbors]
        return
Counter(nearest_neighbors).most_common(1)[0][0] #
cari yg paling banyak

    def predict(self, X_test):
        # X_test itu semua data test
        return [self.predict_instance(instance) for
_, instance in X_test.iterrows()]

# Simpan model
def save_model(self, file_name):
    """Save the model to a file."""
    with open(file_name, 'wb') as file:
        pickle.dump(self, file)
    print(f"Model saved to {file_name}")

# Load model
@staticmethod
def load_model(file_name):
    """Load the model from a file."""
    with open(file_name, 'rb') as file:
        model = pickle.load(file)
    print(f"Model loaded from {file_name}")
    return model

```

2. Implementasi Algoritma Naive Bayes

Gaussian Naive Bayes adalah bagian dari algoritma *Naive Bayes* yang mengasumsikan bahwa fitur-fitur dalam *dataset* sudah mengikuti distribusi *Gaussian* atau normal. Implementasi menggunakan konsep *probability posterior* berdasarkan *Bayes Theorem*.

Implementasi:

1. *Training Model* (`fit`)

- $P(v_j)$: *Prior probability*

$$P(v_j) = \frac{\text{Jumlah data di kelas } v_j}{\text{Jumlah total data}}$$

- Distribusi *Gaussian*:

$$\text{mean} = \frac{\sum x}{N}, \text{std} = \sqrt{\frac{\sum (x - \text{mean})^2}{N}}$$

2. Menghitung *Likelihood* (`_calculate_likelihood`)

$$P(x|v_j) = \frac{1}{\sqrt{2\pi \cdot \sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

3. *Posterior Probability* (`_calculate_posterior`)

$$P(v_j|X) = P(v_j) \cdot P(x_1|v_j) \cdot P(x_2|v_j) \dots P(x_n|v_j)$$

- $P(v_j)$: *prior class*
- $P(x_i|v_j)$: *likelihood*

4. Prediksi

- Menghitung probabilitas posterior untuk semua *class*
- Memilih *Class* dengan posterior paling tinggi sebagai prediksi

5. Implementasi

- Model dapat disimpan menggunakan *pickle*

```
import numpy as np
import pandas as pd
from collections import defaultdict
import pickle

class GaussianNaiveBayes:
    def __init__(self):
        self.priors = {}
        self.mean_std = defaultdict(dict)
        self.classes = None

    def fit(self, X, y):
```

```

        # Pastikan y berbentuk array numpy
        y = np.array(y)
        self.classes = np.unique(y)

        for c in self.classes:
            # Ambil subset data untuk setiap kelas
            X_c = X[y == c]

            # Hitung prior  $P(v_j)$  untuk kelas c
            self.priors[c] = len(X_c) / len(y)

            # Hitung mean dan std dev untuk setiap
            # fitur di kelas c
            self.mean_std[c]['mean'] =
            X_c.mean(axis=0)
            self.mean_std[c]['std'] = X_c.std(axis=0)

        def _calculate_likelihood(self, x, mean, std):
            # Menghitung distribusi Gaussian
            exponent = np.exp(-((x - mean) ** 2) / (2 *
            std ** 2))
            return (1 / (np.sqrt(2 * np.pi) * std)) *
            exponent

        def _calculate_posterior(self, x):
            posteriors = {}

            for c in self.classes:
                # Ambil prior  $P(v_j)$ 
                prior = self.priors[c]

                # Ambil mean dan std dev untuk kelas c
                mean = self.mean_std[c]['mean']
                std = self.mean_std[c]['std']

                # Hitung likelihood  $P(a_1 | v_j) * P(a_2 |
                v_j) * \dots * P(a_n | v_j)$ 
                likelihood =
                np.prod(self._calculate_likelihood(x, mean, std))

                # Hitung posterior  $P(v_j | a_1, a_2, \dots,
                a_n)$ 
                posteriors[c] = prior * likelihood

            return posteriors

```

```

def predict(self, X):
    # Konversi X ke array numpy jika berbentuk
    DataFrame
    X = np.array(X)
    predictions = []
    for x in X:
        # Hitung posterior untuk setiap kelas
        posteriors = self._calculate_posterior(x)
        # Pilih kelas dengan posterior tertinggi
        predictions.append(max(posteriors,
key=posteriors.get))
    return predictions

def save_model(self, file_name):
    """Menyimpan model ke file."""
    with open(file_name, 'wb') as file:
        pickle.dump(self, file)
    print(f"Model saved to {file_name}")

    @staticmethod
    def load_model(file_name):
        """Memuat model dari file."""
        with open(file_name, 'rb') as file:
            model = pickle.load(file)
        print(f"Model loaded from {file_name}")
        return model

```

3. Implementasi Algoritma ID3

ID3 (*Iterative Dichotomiser 3*) adalah algoritma *decision tree learning* yang menggunakan *information gain* untuk membangun *tree* berdasarkan atribut-atribut *dataset*.

Implementasi:

1. Menghitung *Entropy*

Mengukur ketidakpastian *class* dalam *dataset*:

$$H(S) = - \sum_{i=1}^n p_i \log_2(p_i)$$

2. *Information Gain*

Menentukan fitur paling baik untuk memisahkan data:

$$IG(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

3. *Binary Splitting*

Pohon dibangun secara rekursif dengan:

- Mencari feature dan threshold terbaik untuk split menggunakan information gain.
- Membagi dataset ke left node dan right node berdasarkan threshold (binary splitting).
- Menghentikan rekursi jika kedalaman maksimum tercapai, jumlah sampel terlalu kecil, atau data dalam node sudah homogen.

4. Optimisasi

- *Masking* digunakan untuk mem-filter data tanpa membuat copy baru untuk tiap node, sehingga efisien dalam penggunaan memori.
- Split hanya dilakukan jika jumlah sampel cukup besar (`min_samples`) dan information gain melebihi ambang tertentu (`min_gain`).

5. Save Model

Model dapat disimpan menggunakan pickle. Disimpan dengan menggunakan `save_model(<nama file>)` dan di load dengan `load_model(<nama file>)`.

```
import pickle
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from time import time

# Ngitung entropy
def entropy(y):
    if len(y) == 0:
        return 0
    classes, counts = np.unique(y,
return_counts=True)
    probabilities = counts / len(y)
    return -np.sum(probabilities *
np.log2(probabilities + 1e-10))

# Menghitung information gain untuk sebuah fitur dan
threshold tertentu
def information_gain(data, feature, target,
min_samples=50):
    # Skip kalo datanya terlalu dikit
    if len(target) < min_samples:
        return 0, None

    total_entropy = entropy(target)
```



```

# Inisialisasi threshold
best_threshold = None
best_gain = -1

# Cari nilai unik untuk split
unique_values = np.unique(data[feature])

# Kalo cuma ada 1 nilai unique, ga bisa di-split
if len(unique_values) <= 1:
    return 0, unique_values[0] if
len(unique_values) == 1 else None

# Kalo nilai uniknya kebanyakan, ambil sebagian
aja
if len(unique_values) > 10:
    percentiles = np.percentile(data[feature],
[25, 50, 75])
    unique_values =
np.unique(np.concatenate([unique_values[:5],
percentiles, unique_values[-5:]]))

# Iterasi nilai threshold yang mungkin
for threshold in unique_values:
    # Pake mask daripada bikin copy data
    left_mask = data[feature] <= threshold
    right_mask = ~left_mask

    # Skip kalo splitnya ga seimbang
    if np.sum(left_mask) < 2 or
np.sum(right_mask) < 2:
        continue

    # Hitung entropy masing-masing split
    left_entropy = entropy(target[left_mask])
    right_entropy = entropy(target[right_mask])

    # Hitung bobot berdasarkan jumlah sampel
    weight_left = np.sum(left_mask) / len(target)
    weight_right = np.sum(right_mask) /
len(target)

    # Hitung entropy setelah split
    split_entropy = weight_left * left_entropy +
weight_right * right_entropy

```

```

        # Menghitung information gain
        gain = total_entropy - split_entropy

        # Memperbarui threshold terbaik jika gain
        lebih baik
        if gain > best_gain:
            best_gain = gain
            best_threshold = threshold

    return best_gain, best_threshold

class ID3:
    def __init__(self, max_depth=None,
min_samples=50, min_gain=1e-4):
        self.tree = None
        self.max_depth = max_depth
        self.min_samples = min_samples
        self.min_gain = min_gain
        self.label_encoder = LabelEncoder()
        self.n_samples = 0
        self.start_time = None
        self.message = "ID3 Numeric Classifier"

    # Membangun decision tree secara rekursif
    def fitter(self, X, y, depth=0):
        n_samples = len(y)

        # Print progress tiap beberapa level
        if depth % 2 == 0:
            elapsed = time() - self.start_time
            progress = (self.n_samples - n_samples) /
self.n_samples * 100
            print(f"Level {depth}, Progress:
{progress:.1f}%, "
                  f"Samples: {n_samples}, Time:
{elapsed:.1f}s")

        # Base cases - bikin leaf node
        if (self.max_depth is not None and depth >=
self.max_depth) or \
            n_samples < self.min_samples or \
            len(np.unique(y)) == 1:
            return np.bincount(y).argmax()

    # Inisialisasi pencarian split terbaik

```

```

        best_feature = None
        best_threshold = None
        best_gain = -1

        # Cari feature dan threshold terbaik
        for feature in X.columns:
            gain, threshold = information_gain(X,
feature, y, self.min_samples)
            if gain > best_gain and gain >
self.min_gain:
                best_gain = gain
                best_feature = feature
                best_threshold = threshold

        # Kalo ga ketemu split yang bagus, jadiin
leaf node
        if best_gain <= self.min_gain or
best_threshold is None:
            return np.bincount(y).argmax()

        # Bikin split pake mask
        left_mask = X[best_feature] <= best_threshold
        right_mask = ~left_mask

        # Cek split identik
        if np.sum(left_mask) == 0 or
np.sum(right_mask) == 0:
            return np.bincount(y).argmax()

        # Bikin node dan rekursi ke child nodes
        return {
            'feature': best_feature,
            'threshold': best_threshold,
            'left': self.fitter(X[left_mask],
y[left_mask], depth + 1),
            'right': self.fitter(X[right_mask],
y[right_mask], depth + 1)
        }

    def fit(self, X, y):
        # Encode label kalo bukan numerik
        self.n_samples = len(y)
        self.start_time = time()
        print(f"Mulai training dengan
{self.n_samples} samples...")

```

```

        # Transform label jadi numerik
        y_encoded =
self.label_encoder.fit_transform(y)

        # Build tree
        self.tree = self.fitter(X, y_encoded)
        print(f"Training selesai dalam {time() -
self.start_time:.1f} detik")

        # Prediksi kelas untuk satu instance
        def predict_instance(self, instance, tree):
            if not isinstance(tree, dict):
                return tree

            if instance[tree['feature']] <=
tree['threshold']:
                return self.predict_instance(instance,
tree['left'])
            else:
                return self.predict_instance(instance,
tree['right'])

        # Prediksi kelas untuk banyak instances
        def predict(self, X):
            numeric_predictions =
[self.predict_instance(row, self.tree)
                                for _, row in
X.iterrows()]
            # Kembalikan ke label asli
            return
self.label_encoder.inverse_transform(numeric_predicti
ons)

        # Simpan model
        def save_model(self, file_name):
            with open(file_name, 'wb') as file:
                pickle.dump(self, file)
            print(f"Model saved to {file_name}")

        # Load model
        @staticmethod
        def load_model(file_name):
            with open(file_name, 'rb') as file:
                model = pickle.load(file)
            print(f"Model loaded from {file_name}")
            return model

```

BAB 2

Tahap *Cleaning* dan *Preprocessing*

1. *Data Cleaning*

- *Handling Missing Data*

Missing values pada data numerik di-handle dengan imputasi mean atau median karena mean dan median efektif untuk data numerik yang tidak memiliki banyak *outlier* sedangkan fitur kategori menggunakan mode karena untuk mempertahankan nilai yang paling umum. Implementasi kelas `FeatureImputer` untuk *handle* kolom dengan nilai kosong yang melebihi batas tertentu.

- *Handling Outliers*

Untuk memotong nilai *outlier* di luar batas yang ditentukan digunakan teknik *clipping* dengan IQR (*Interquartile Range*) karena *outlier* dapat merusak performa algoritma yang *distance based* seperti KNN dan *clipping* untuk mempertahankan data asli sambil membatasi akibat dari *outlier* tersebut.

- *Remove Duplicates*

Data yang duplikat dihapus berdasarkan kolom id dengan menggunakan `drop_duplicates` karena duplikat bisa menyebabkan bias dan mempengaruhi performa model. Implementasi kelas `DuplicateImputer` untuk mengidentifikasi dan menghapus data yang duplikat.

2. *Data Preprocessing*

- *Scaling*

Menggunakan *Min-Max Scaling* untuk meratakan *range* nilai numerik ke dalam skala [0,1] karena algoritma KNN dan *Naive Bayes* sensitif terhadap skala data dan *Scaling* memastikan semua fitur memiliki kontribusi yang seimbang. Implementasi kelas `FeatureScaler` untuk menghitung nilai minimum dan maksimum dari semua fitur yang numerik.

- *Encoding*

Menggunakan *One-Hot Encoding* di fitur kategori untuk mengubah menjadi format numerik karena algoritma *machine learning* secara umum bekerja dengan data yang numerik dan *One-Hot Encoding* mempertahankan nilai unik dari fitur kategori tanpa mengasumsikan urutan. Implementasi kelas `FeatureEncoder` untuk *encode* fitur *categorical*.

- *Handling Imbalanced Data*

Menggunakan SMOTE (*Synthetic Minority Oversampling Technique*) untuk menyeimbangkan distribusi kelas karena distribusi kelas pada *dataset* tidak seimbang, seperti misalnya kelas *normal* dan *attack* berbeda signifikan dan

SMOTE efektif untuk menambah sampel kelas yang minoritas tanpa kehilangan informasi. Implementasi kelas `ImbalancedDataHandler` untuk melakukan *oversampling* menggunakan SMOTE.

- *Dimensionality Reduction*

Menggunakan PCA (*Principal Component Analysis*) untuk mengurangi dimensi fitur sambil mempertahankan variasi data karena *dataset* yang punya fitur terlalu banyak memiliki risiko *overfitting* dan PCA membantu mengurangi fitur redundan dan menyederhanakan data. Implementasi kelas `PCAImputer` untuk melakukan reduksi dimensi dengan menghitung komponen utama.

3. *Pipeline Preprocessing*

Pipeline mengombinasikan semua langkah di atas untuk memastikan proses *cleaning* dan transformasi data berjalan dengan baik. *Pipeline* memastikan data siap untuk digunakan sebagai model *machine learning*.

Langkah-langkah *pipeline*:

1. *Outlier Handling* (`OutlierImputer`)
2. *Missing Value Imputation* (`FeatureImputer`)
3. *Duplicate Removal* (`DuplicateImputer`)
4. *Scaling dan Encoding* (`FeatureScaler` dan `FeatureEncoder`)
5. *Dimensionality Reduction* (`PCAImputer`)
6. *Scaling* (`FeatureScaler`)
7. *Handling Imbalanced Data* (SMOTE)

Dengan menangani *missing values*, *outliers*, dan duplikat bisa menjaga kebersihan data dan integritas *dataset*. Selain itu, *scaling* membuat algoritma jarak bekerja lebih baik. *Dimensionality reduction* membantu mengurangi kekompleksitasan model dan kemudian dilakukan *scaling* lagi untuk menyesuaikan hasil PCA. *Handling imbalanced data* ditujukan untuk membuat performa model seimbang untuk semua kelas. Sehingga algoritma seperti KNN, ID3, dan *Naive Bayes* dapat bekerja optimal dan menghasilkan prediksi yang akurat.

BAB 3

Hasil

1. Hasil Algoritma Naive Bayes

```
Naive Bayes From Scratch

from src.models.NaiveBayes import GaussianNaiveBayes
from sklearn.metrics import accuracy_score

model = GaussianNaiveBayes()
model.fit(x_train_final, y_train_final)
model.save_model("NaiveBayesFromScratch.pkl")

prediction = model.predict(x_val_final)
accuracy = accuracy_score(y_val_final, prediction)
print("Accuracy (Naive Bayes From Scratch): ", accuracy)

Model disimpan ke NaiveBayesFromScratch.pkl
Accuracy (Naive Bayes From Scratch): 0.12594712455710003
```

Gambar hasil dengan Algoritma Naive Bayes from Scratch

```
Naive Bayes From Sklearn

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Train the model
model_sklearn = GaussianNB()
model_sklearn.fit(x_train_final, y_train_final)
prediction = model_sklearn.predict(x_val_final)
accuracy = accuracy_score(y_val_final, prediction)
print("Accuracy (Sklearn Naive Bayes): ", accuracy)

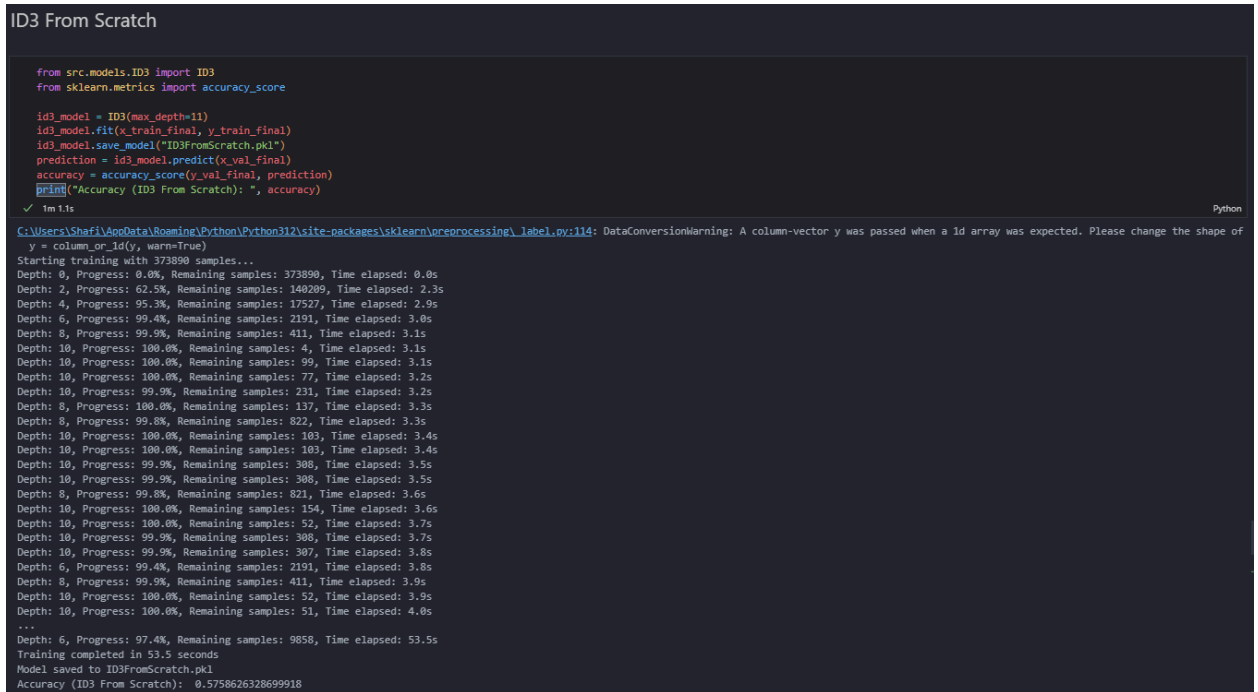
C:\Users\Shafii\AppData\Local\Temp\Python312\site-packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to
y = column_or_1d(y, warn=True)
Accuracy (Sklearn Naive Bayes): 0.4701662578359226
```

Gambar hasil dengan Algoritma Naive Bayes from Sklearn

Hasil implementasi kami menunjukkan akurasi yang jauh lebih rendah (0.125) dibandingkan dengan implementasi Scikit-learn (0.470). Perbedaan ini terjadi karena beberapa hal. Pertama, algoritma dari scratch cenderung kurang optimal dalam menangani kasus-kasus khusus, seperti standar deviasi nol atau data dengan nilai unik yang terbatas. Selain itu, penggunaan metode precomputed likelihoods pada implementasi scratch kurang cocok untuk data kontinu, sehingga perhitungan probabilitasnya menjadi kurang akurat.

Selain itu, Scikit-learn lebih unggul karena menggunakan optimisasi berbasis Cython. Teknologi ini tidak hanya membuat proses komputasi lebih cepat, tapi juga memastikan algoritmanya lebih stabil dan matang. Jadi, meskipun implementasi manual ini adalah langkah yang bagus untuk memahami algoritma Naive Bayes, hasilnya memang belum bisa menyamai efisiensi dan akurasi dari Scikit-learn.

2. Hasil Algoritma ID3



```
from src.models.ID3 import ID3
from sklearn.metrics import accuracy_score

id3_model = ID3(max_depth=11)
id3_model.fit(x_train_final, y_train_final)
id3_model.save_model("ID3FromScratch.pkl")
prediction = id3_model.predict(x_val_final)
accuracy = accuracy_score(y_val_final, prediction)
print("Accuracy (ID3 From Scratch): ", accuracy)
```

✓ 1m 11s

C:\Users\Shafil\AppData\Local\Programs\Python\Python312\site-packages\sklearn\preprocessing_label.py:114: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y = column or 1d(y, warn=True)

Starting training with 373890 samples...

Depth: 0, Progress: 0.0%, Remaining samples: 373890, Time elapsed: 0.0s

Depth: 2, Progress: 62.5%, Remaining samples: 148209, Time elapsed: 2.3s

Depth: 4, Progress: 95.3%, Remaining samples: 17527, Time elapsed: 2.9s

Depth: 6, Progress: 99.4%, Remaining samples: 2191, Time elapsed: 3.0s

Depth: 8, Progress: 99.9%, Remaining samples: 411, Time elapsed: 3.1s

Depth: 10, Progress: 100.0%, Remaining samples: 4, Time elapsed: 3.1s

Depth: 10, Progress: 100.0%, Remaining samples: 99, Time elapsed: 3.1s

Depth: 10, Progress: 100.0%, Remaining samples: 77, Time elapsed: 3.2s

Depth: 10, Progress: 99.9%, Remaining samples: 231, Time elapsed: 3.2s

Depth: 8, Progress: 100.0%, Remaining samples: 137, Time elapsed: 3.3s

Depth: 8, Progress: 99.8%, Remaining samples: 822, Time elapsed: 3.3s

Depth: 10, Progress: 100.0%, Remaining samples: 103, Time elapsed: 3.4s

Depth: 10, Progress: 100.0%, Remaining samples: 103, Time elapsed: 3.4s

Depth: 10, Progress: 99.9%, Remaining samples: 308, Time elapsed: 3.5s

Depth: 10, Progress: 99.9%, Remaining samples: 308, Time elapsed: 3.5s

Depth: 8, Progress: 99.8%, Remaining samples: 821, Time elapsed: 3.6s

Depth: 10, Progress: 100.0%, Remaining samples: 154, Time elapsed: 3.6s

Depth: 10, Progress: 100.0%, Remaining samples: 52, Time elapsed: 3.7s

Depth: 10, Progress: 99.9%, Remaining samples: 308, Time elapsed: 3.7s

Depth: 10, Progress: 99.9%, Remaining samples: 307, Time elapsed: 3.8s

Depth: 6, Progress: 99.4%, Remaining samples: 2191, Time elapsed: 3.8s

Depth: 8, Progress: 99.9%, Remaining samples: 411, Time elapsed: 3.9s

Depth: 10, Progress: 100.0%, Remaining samples: 52, Time elapsed: 3.9s

Depth: 10, Progress: 100.0%, Remaining samples: 51, Time elapsed: 4.0s

...

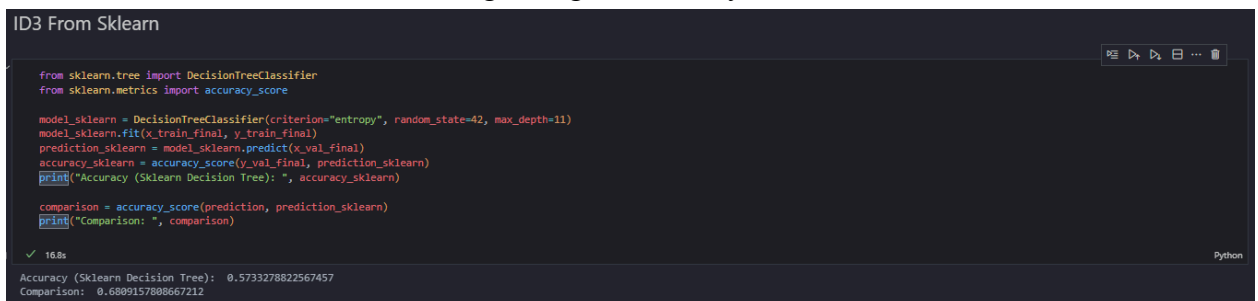
Depth: 6, Progress: 97.4%, Remaining samples: 9858, Time elapsed: 53.5s

Training completed in 53.5 seconds

Model saved to ID3FromScratch.pkl

Accuracy (ID3 From Scratch): 0.5758626328699918

Gambar hasil dengan Algoritma *ID3 from Scratch*



```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

model_sklearn = DecisionTreeClassifier(criterion="entropy", random_state=42, max_depth=11)
model_sklearn.fit(x_train_final, y_train_final)
prediction_sklearn = model_sklearn.predict(x_val_final)
accuracy_sklearn = accuracy_score(y_val_final, prediction_sklearn)
print("Accuracy (Sklearn Decision Tree): ", accuracy_sklearn)

comparison = accuracy_score(prediction, prediction_sklearn)
print("Comparison: ", comparison)
```

✓ 16.8s

Accuracy (Sklearn Decision Tree): 0.5733278822567457

Comparison: 0.6809157088667212

Gambar hasil dengan Algoritma ID3 Sklearn

Dalam implementasi algoritma ID3, proses perhitungan information gain untuk seleksi fitur dilakukan menggunakan metode binary split. Selain itu, pembentukan tree tidak membuat salinan baru pada setiap node, melainkan memanfaatkan masking untuk mem-filter data. Pendekatan ini terbukti dapat mempercepat proses pembuatan model.

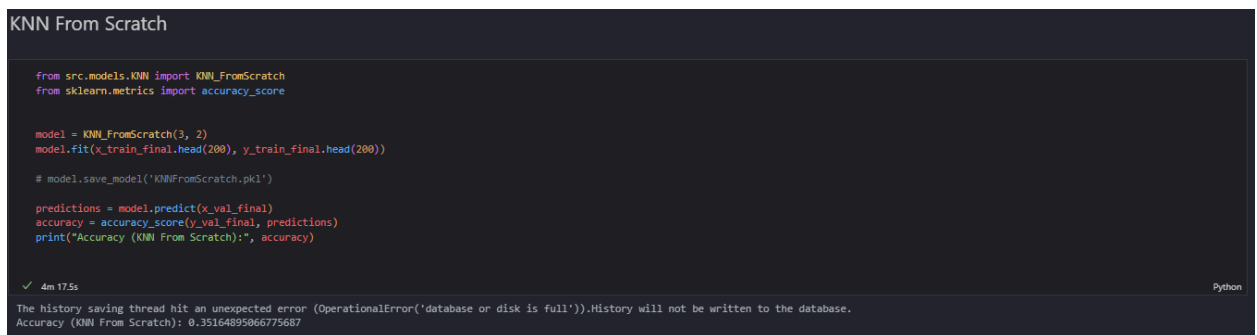
Namun, jika dibandingkan dengan Scikit-learn, waktu eksekusi implementasi kami jauh lebih lambat (Sckit-learn: 16.8 detik , Implementasi kami: 61.1 detik). Hal ini disebabkan oleh Scikit-learn yang menggunakan Cython, sebuah teknologi yang mengkompilasi Python menjadi C untuk meningkatkan performa, terutama pada komputasi berat.

Dari segi akurasi, implementasi kami menunjukkan hasil yang sedikit lebih baik (accuracy = 0.5758626328699918) dibandingkan Scikit-learn (accuracy =

0.5733278822567457) pada maximum depth tree = 11. Kedalaman tree dipilih hingga 11 agar prediksi dapat mencakup semua kelas. Hasil akurasi yang lebih baik ini kemungkinan karena dataset training yang digunakan lebih cocok dengan metode splitting pada algoritma ID3 kami. Selain itu, faktor seperti metode validasi yang kurang robust juga mungkin mempengaruhi hasil. Secara keseluruhan, hasil yang sama antara implementasi kami dan Sklearn itu 68%.

Perlu dicatat bahwa algoritma Decision Tree di Scikit-learn bukan ID3, melainkan CART (Classification and Regression Tree). CART dirancang lebih efektif dalam menangani berbagai jenis data, termasuk data kontinu, serta mendukung pruning untuk mencegah overfitting.

3. Hasil Algoritma KNN



```
KNN From Scratch

from src.models.KNN import KNN_FromScratch
from sklearn.metrics import accuracy_score

model = KNN_FromScratch(3, 2)
model.fit(x_train_final.head(200), y_train_final.head(200))

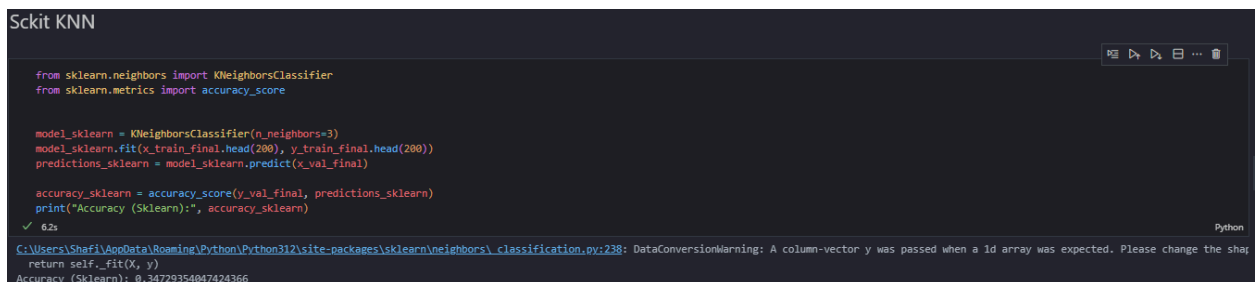
# model.save_model('KNNFromScratch.pkl')

predictions = model.predict(x_val_final)
accuracy = accuracy_score(y_val_final, predictions)
print("Accuracy (KNN From Scratch):", accuracy)

✓ 4m 17.5s Python

The history saving thread hit an unexpected error (OperationalError('database or disk is full')).History will not be written to the database.
Accuracy (KNN From Scratch): 0.35164895066775687
```

Gambar hasil dengan Algoritma *KNN from Scratch*



```
Sckit KNN

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

model_sklearn = KNeighborsClassifier(n_neighbors=3)
model_sklearn.fit(x_train_final.head(200), y_train_final.head(200))
predictions_sklearn = model_sklearn.predict(x_val_final)

accuracy_sklearn = accuracy_score(y_val_final, predictions_sklearn)
print("Accuracy (Sklearn):", accuracy_sklearn)

✓ 6.2s Python

C:\Users\Shaf1\AppData\Roaming\Python\Python312\site-packages\sklearn\neighbors\_classification.py:238: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,).
return self._fit(X, y)
Accuracy (Sklearn): 0.34729354047424366
```

Gambar hasil dengan Algoritma KNN Sklearn

Pada implementasi KNN, perhitungan distance setiap baris terlalu lama jadi pada tangkapan layar di bawah hanya memakai 200 baris data. Untuk memilih tipe distance yang digunakan menggunakan parameter p untuk rumus Minkowski distance. p dapat bernilai 1 untuk Manhattan distance, 2 untuk Euclidean distance, maupun nilai yang lain.

Hasil perbandingan antara KNN From Scratch dan Scikit-learn KNN menunjukkan bahwa akurasi kedua metode cukup mirip, dengan KNN From Scratch sedikit lebih unggul (0.3516 dibandingkan 0.3473). Perbedaan kecil ini kemungkinan disebabkan oleh cara implementasi algoritma yang berbeda. Dalam KNN From Scratch, perhitungan jarak menggunakan fungsi manual dengan parameter Minkowski distance, yang bisa menghasilkan nilai jarak sedikit berbeda dibandingkan Scikit-learn.

Namun, dari segi waktu eksekusi, Scikit-learn jauh lebih cepat, dengan waktu hanya 62 detik dibandingkan implementasi manual yang memakan waktu hingga 4 menit 17 detik. Perbedaan ini terjadi karena Scikit-learn menggunakan Cython dan struktur data seperti KD-Tree atau Ball-Tree untuk mempercepat pencarian tetangga terdekat, sementara implementasi *from scratch* menggunakan Python murni yang jelas lebih lambat.

KONTRIBUSI

NIM	Nama	Tugas
13521117	Maggie Zeta RS	Laporan, Testing
13522003	Shafiq Irfansyah	ID3, Naive Bayes, Testing
13522064	Devinzen	KNN, Testing
13522107	Rayendra Althaf TN	Preprocessing, Testing
13522154	Chelvadinda	Naive Bayes, Testing

REFERENSI

- https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250331293_IF3170_Materi09_Seg01_AI-kNN.pdf
- https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250380758_IF3170_Materi09_Seg02_AI-NaiveBayes.pdf
- https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250430397_IF3170_Materi09_Seg03_AI-PredictionMeasurement.pdf
- https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/297373-Data-Preparation/1730818790714_IF-3170-Data-Preparation.pdf